Improving Disk Performance: A Prefetching Scheme Exploiting Data Layout and Access History

Kei Davis, CCS-7;
Xiaoning Ding,
Xiaodong Zhang, The Ohio State University
Song Jiang, Wayne State University

Prefetching is an important technique for improving effective hard disk performance. It attempts to accurately predict the data to be requested and load it ahead of the arrival of application requests. Current disk prefetch policies in major operating systems (OS) track access patterns at the level of file abstraction, which has limitations and so cannot realize the full performance improvements achievable by prefetching. We have designed a system, DiskSeen, that performs prefetching directly at the level of disk layout, and in a portable way. An important design consideration is that our technique is entirely supplementary to, and works synergistically with, any present file-level prefetch policies. Our implementation in the Linux kernel shows that it can significantly improve the effectiveness of prefetching, reducing execution times by 20–53% for micro-benchmarks and real applications. Even with workloads specifically designed to expose its weaknesses DiskSeen exhibits only minor performance loss.

Current disk prefetch policies in major OSs track access patterns at the level of file abstraction. While this is useful for exploiting application-level access patterns, there are two reasons why file-level prefetching cannot realize the full performance improvements achievable by prefetching: (1) certain prefetch opportunities can only be detected by knowing the data layout on disk, such as the contiguous layout of file meta-data or data from multiple files; and (2) non-sequential access of disk data (requiring disk head movement) is much slower than sequential access, and the penalty for mis-prefetching a "random" block (unit of disk capacity), relative to that of a sequential block, is correspondingly more costly. To overcome the inherent limitations of prefetching at the logical file level we propose to perform prefetching directly at the level of disk layout and in a portable way. Our technique, called DiskSeen, is intended to be supplementary to, and to work synergistically with, any present file-level prefetch policies.

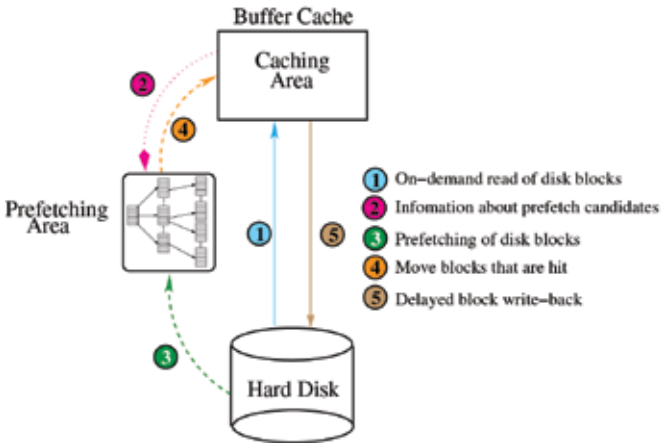

Fig. 1. A simplified diagram of the DiskSeen system.

In essence DiskSeen is a sequence-based, history-aware prefetch scheme based on two observations: (1) accesses of disk blocks in a particular order are likely to be repeated; and (2) because it is performed asynchronously with application execution, if prefetching is not so inaccurate as to interfere with application progress, it is essentially free.

DiskSeen tracks the locations and access times of disk blocks, and based on analysis of their temporal and spatial relationships, seeks to improve the sequentiality of disk accesses and overall prefetching performance. It also implements a mechanism to minimize mis-prefetching to mitigate the corresponding performance penalty.

We leave file-level prefetching enabled; DiskSeen concurrently performs prefetching at a lower level to mitigate the inadequacies of file-level prefetching. DiskSeen seeks to detect sequences of block accesses based on block disk addresses, or logical block numbers (LBN). At the same time, it maintains block access history and uses the history information to further improve the effectiveness of prefetching when recorded access patterns are observed to be repeated.

There are four objectives in the design of DiskSeen: (1) Efficiency–we ensure that prefetched blocks are in a localized disk area and are accessed in ascending order of their LBNs for optimal disk performance; (2) Eagerness–prefetching is initiated immediately when a prefetching opportunity emerges; (3) Accuracy–only the blocks that are highly likely to be requested are prefetched–significant inaccurate prefetching automatically suppresses prefetching; and (4) Aggressiveness–prefetching is made more aggressive if it helps with accuracy and reduces request service times. Conversely, if it is detected to be increasing service times (because of inaccurate predictions) it is throttled back.

Buffer cache is divided into prefetching and caching areas according to their roles in the scheme (see Fig. 1). A block could be prefetched into the prefetching area based on either current or historical access information—both are recorded in the disk block table, or as directed by

For more information contact Kei Davis at kei@lanl.gov.

Funding Acknowledgments
DOE, NNSA, Advanced Simulation and Computing Program

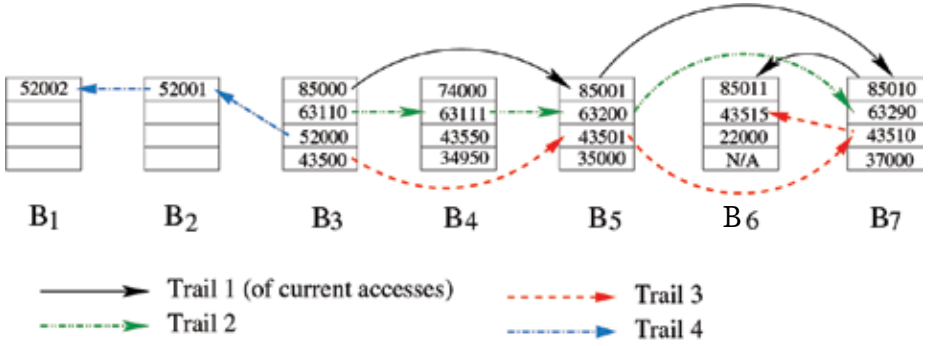

Fig. 2. Access trails. B1 through B7 are consecutive contiguous blocks in the block table. There are four trails starting from block B3: one current trail and three history trails. Trail 1 (B3, B5, B7, B6) corresponds to the on-going continuous block accesses.

file-level prefetching. The caching area corresponds to the traditional buffer cache and is managed by the existing OS kernel policies except that prefetched but not-yet-requested blocks are no longer stored in the cache. A block is read into the caching area either from the prefetching area, if it is hit there, or directly from disk, all in an on-demand fashion.

A central component of DiskSeen is its mechanism for maintaining access sequence history. To describe access history we introduce the term trail to describe a sequence of blocks that has been accessed with a small time interval between each consecutive pair of blocks in the sequence and is located in a bounded region. DiskSeen maintains a history of previously seen access sequences—trails—to provide its predictive capability. An important point is that a trail may be any sequence of blocks, whereas conventional disk-level prefetching typically relies the detection of strictly sequential access of contiguous blocks. Figure 2 depicts a segment of the trail history data structure.

Figure 3 shows the execution times for a selection of benchmarks (described in [1]) chosen both to showcase DiskSeen and to specifically thwart DiskSeen's predictive capability. Two observations are that (1) even when DiskSeen is wholly ineffective, it does not significantly hurt performance; and (2) an application need not necessarily be run more than once for the history mechanism to be useful.

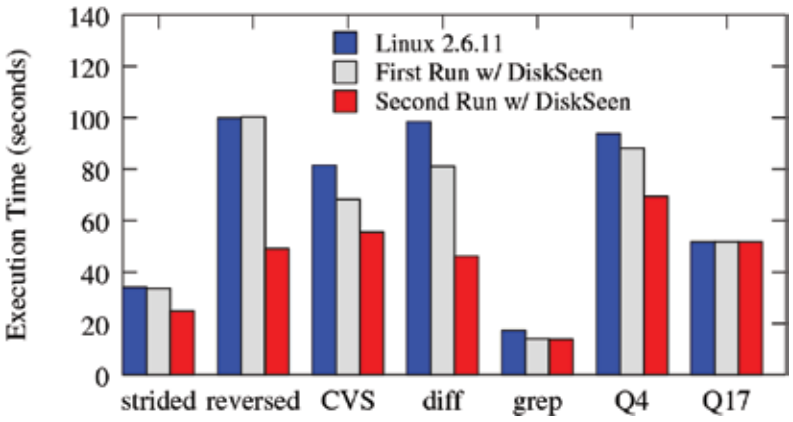

Fig. 3. Execution times of six benchmarks. Linux 2.6.11 refers to the stock Linux kernel.

[1] Jiang, S.. et al., "A Prefetching Scheme Exploiting both Data Layout and Access History on Disk," *ACM Trans Storage*, to appear (2013).